Yan Liu

Compte Rendu TD07

-Implémentation du TD06

Une fonction qui retourne la property du Noeud: get_prop_val(id prop)

```
(defun get_prop_val (id prop)
    (cdr (assoc prop (symbol-value id))) );;symbol-value get
valeur du id (comme eval)
```

```
Ex:  N1=> ( (NAME . LIU) (AGE . 23) )
     get_prop_val('N1 'name) => LIU
```

Une fonction qui peut set property: set_prop_val(id prop val)

```
(defun set_prop_val (id prop val)
    (setf
       (cdr (assoc prop (symbol-value id)));; peut pas
directement appele get_prop_val()
        val
    )
)
```

```
Ex: (set_prop_val 'N1 'name 'LIUYan) => ( (NAME . LIUYan) (AGE
     . 23) )
```

Une fonction qui peut ajouter un nouveau property est set une valeur: add_prop_val (id prop val)

```
(defun add_prop_val (id prop val)
;;ajoute la valeur val a prop
    (let ((pair (assoc prop (symbol-value  id))))
        (if pair   ;; tester si prop est exist.
            (set_prop_val id prop (cons val (cdr pair)))
            (push (cons prop val) (symbol-value id))
        )
    )
)
```

```
Ex:(add_prop_val 'N1 'pay ‘Chine)=>( (PAY . Chine) (NAME .
LIUYan) (AGE . 23) )
```

-TD07

Écrivez les fonctions LISP implémentant le mécanisme de propagation de marques. Outre un ensemble de fonctions de service, vous définirez les fonctions principales sui- vantes :
– mark-node : marque un nœud avec une marque identifiée.

– wave : progage une marque le long des arcs d’un certain type, dans le sens direct ou contraire.

– get-results : permet d’interroger les résultats.

```
(defun mark_node (id mark)
    (add_prop_val id 'mark mark)
)
```

```
(defun get_marked_nodes (mark)
  ; pour obtenir les noeuds marqué selon mark dans le réseau
    (let ((res))
        (dolist (x *noeuds* (reverse res))
            (if (eq (get 'mark x) mark)
                (push x res)
            )
        )
    )
)
```

Quelques fonction utils:

```
function : marked?(node mark)     successors(node type-arc)
pred(node type-arc)
```

```
(defun marked? (node mark)
  ; pour savoir si marqué selon mark
  (eq (get 'marque node) mark))

(defun pred (node prop)
  ; pour obtenir les prédecesseurs
  (let (res)
    ; on parcourt la liste des arcs x entrant
        (dolist (x (cdr (assoc 'INARCS (symbol-value node)))
(reverse res))
        ; si l'arc x a est du type prop recherché
        (if (eq (cdr (assoc 'TYPE (symbol-value x))) prop)
            ; on retient le nœud d'origine de l'arc (un
prédecesseur)
            (push (cdr (assoc 'FROM (symbol-value x))) res)
            )
        )
    )
)

(defun succ (node prop)
  ; pour obtenir les successeurs
  (let (res)
    ; on parcourt la liste des arcs x entrant
    (dolist (x (cdr (assoc 'OUTARCS (symbol-value node)))
(reverse res))
      ; si l'arc x a est du type prop recherché
      (if (eq (cdr (assoc 'TYPE (symbol-value x))) prop)
        ; on ajoute le nœud destination de l'arc (un
successeur)
        (push (cdr (assoc 'TO (symbol-value x))) res)
      )
    )
  )
)
```

```
;;--------------------version  Dommique,L
;; mark_nodes <- get_marked_nodes(mark)
;; tant que mark_nodes !=nil
;;     new_marked_nodes <- nil
;;     pour chaque noeud de marked_nodes
;;         si direction = direct
;;             next-noeude <-success (noeud type-arc)
;;         sinon
;;             nex-noeuds <- predecesses (noeud type-arc)
;;         finsi
;;         si next-noeuds != nil
;;             pour chaque n de next-noeuds
;;                 si n non marqué par mark
;;                     marquer n
;;                     new_marked_nodes <- new_marked_nodes +n
;;                 finsi
;;             finpour
;;         finsi
;;     finpour
;;     mark_nodes <-new_marked_nodes
;; fintantque
```

Réalisé:

```
(defun wave (mark prop sens)
  ; pour propager les marques
  (if (or (eq sens 'in) (eq sens 'out))
    (let (
        (marked-nodes (get-marked-nodes mark)) ; noeud déjà
marqués
        next-node ; prochains noeuds
        new-marked-node) ; noeuds à marquer
        (loop
            (setq new-marked-node nil) ;
            (dolist (node marked-nodes) ; pour chaque noeuds
marqués
              (setq next-node
                (if (eq sens 'in)
```

```
                (succ node prop) ; on récupère ces
successeurs
                (pred node prop))) ; on récupère ces
prédecesseurs
              (when next-node ; si la liste n'est pas vide
                  (dolist (x next-node)
                    (unless (marked? x mark)
                      (mark-node x mark)
                      (pushnew x new-marked-node))))
            )
            (format t "~& ~A -> ~A" marked-nodes new-marked-
node)
            (unless new-marked-node ; s'il n'y a pas de
nouveaux noeuds à traiter : on arrête
              (return-from wave t))
            (setq marked-nodes new-marked-node)
        )
    ) ; on attribue les nouveaux noeuds
    (format t "~& Erreur : sens mal entré : ~A" sens))
)
```